The GoLand Blog

A Clever IDE to Go

Tutorials

How to Use go:embed in Go 1.16

Florin Pățan

June 9, 2021

One of the most anticipated features of Go 1.16 is the support for embedding files and folders into the application binary at compile-time without using an

external tool. This feature is also known as go:embed, and it gets its name from the compiler directive that makes this functionality possible: //go:embed.

With it, you can embed all web assets required to make a frontend application work. The build pipeline will simplify since the embedding step does not require any additional tooling to get all static files needed in the binary. At the same time, the deployment pipeline is predictable since you don’t

need to worry about deploying the static files and the problems that come with that, such as: making sure the relative paths are what the binary expects, the working directory is the correct one, the application has the proper permissions to read the files, etc. You just deploy the application binary and start it, and everything else works.

Let's see how we can use this feature to our advantage with an example webserver:

- First, create a new Go modules project in

GoLand, and make sure you use Go 1.16 or newer. The go directive in the go.mod file must be set to Go 1.16 or higher too.

```
module goembed.demo

go 1.16
```

- Our main.go file should look like this:

```
package main

import (
```

```
    "embed"
    "html/template"
    "log"
    "net/http"
)

var (
    //go:embed resources
    res embed.FS

    pages = map[string]string{
        "/": "resources/index.gohtml",
    }
```

```
)

func main() {
    http.HandleFunc("/", func(w
http.ResponseWriter, r *http.Request) {
        page, ok := pages[r.URL.Path]
        if !ok {

w.WriteHeader(http.StatusNotFound)
            return
        }
        tpl, err :=
template.ParseFS(res, page)
```

```
        if err != nil {
            log.Printf("page %s not
found in pages cache...", r.RequestURI)

w.WriteHeader(http.StatusInternalServerE
rror)
            return
        }

        w.Header().Set("Content-Type",
"text/html")
        w.WriteHeader(http.StatusOK)
        data := map[string]interface{}{
```

```
            "userAgent": r.UserAgent(),
        }
        if err := tpl.Execute(w, data);
err != nil {
            return
        }
    })

    http.FileServer(http.FS(res))

    log.Println("server started...")
    err := http.ListenAndServe(":8088",
nil)
```

```
    if err != nil {
        panic(err)
    }
}
```

- Next, create a new resources/index.gohtml file like the one below:

```
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>go:embed demo</title>
</head>
```

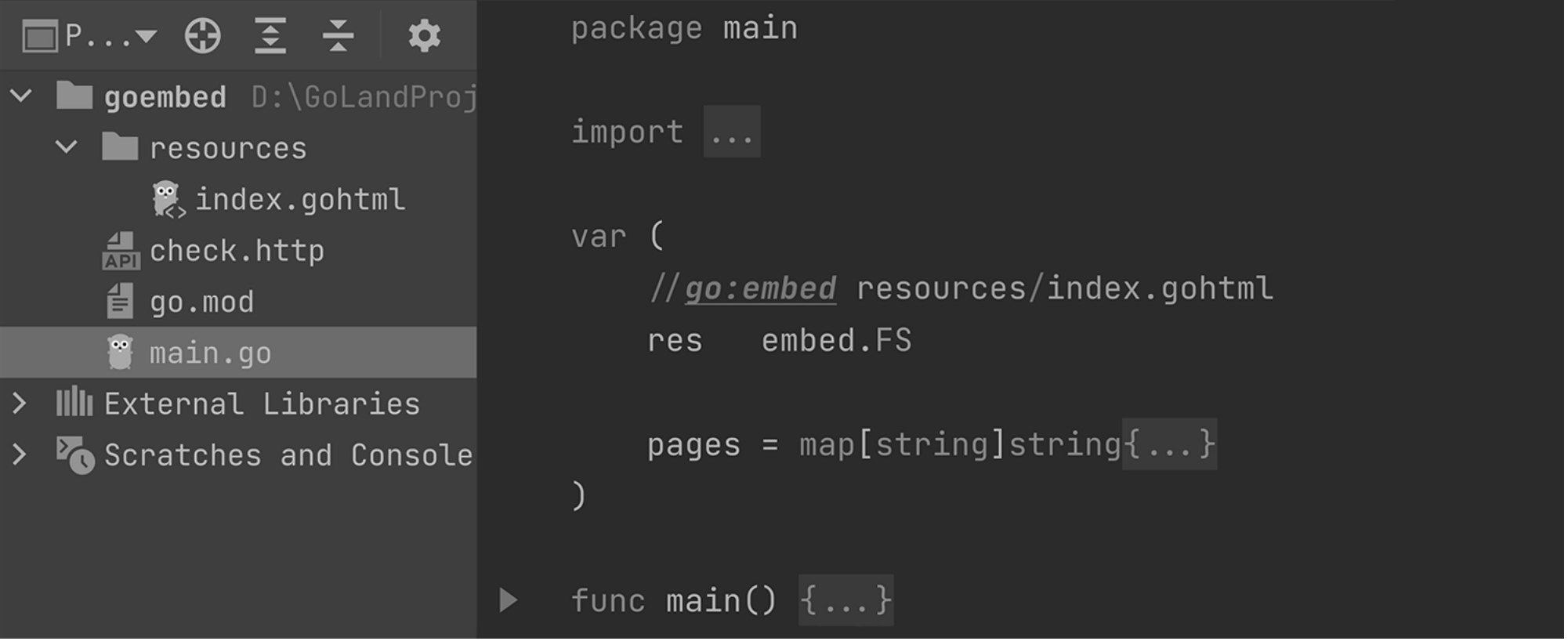
```
<body>
<div>
    <h1>Hello, {{ .userAgent }}!</h1>
    <p>If you see this, then go:embed
worked!</p>
</div>
</body>
</html>
```

- Finally, create a file called check.http at the root of the project. This will reduce the time it takes to test our code by making repeatable requests from GoLand rather than using the browser.

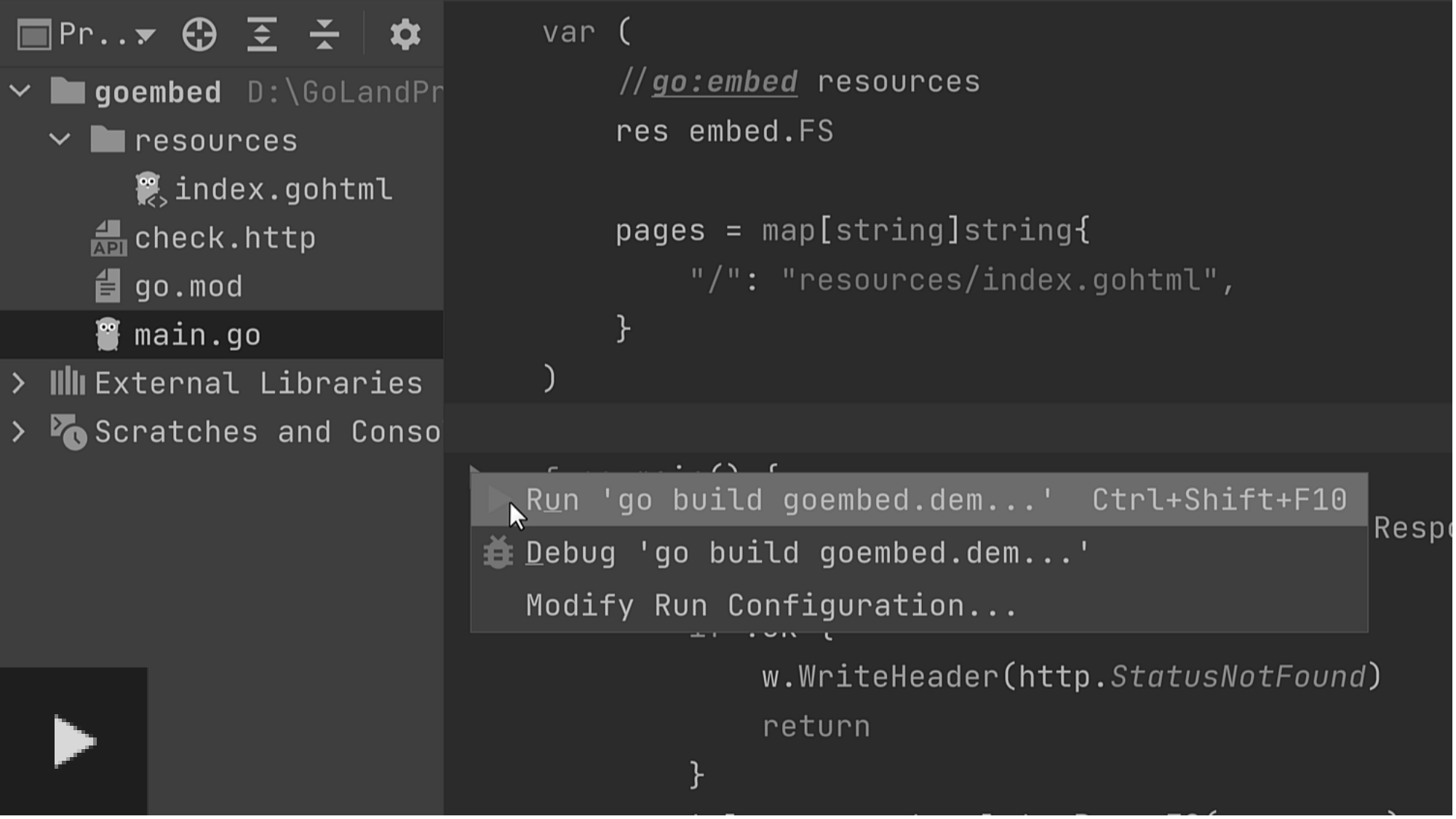
```
GET http://localhost:8088/
```

Note: If you need to, you can download a newer version of Go using GoLand either while creating the project or via Settings/Preferences | Go | GOROOT | + | Download ...

This is how the project layout should look:



If we run this project, then test the request from the check.http file against our server, we get what we'd expect: an HTML response that contains our Hello message and the "Apache-HttpClient" user agent.



At first, this might not look different than any other

server responding to a request with our template code.

However, if we change the code in the template without restarting the server, we'll quickly notice that our output will not change unless we rebuild the binary. We can even remove the template, move the binary, or change its running directory, and the result will be similar. How does this work, then?

A quick look at how go:embed works

We can isolate a few parts of our code that are involved in using this feature.

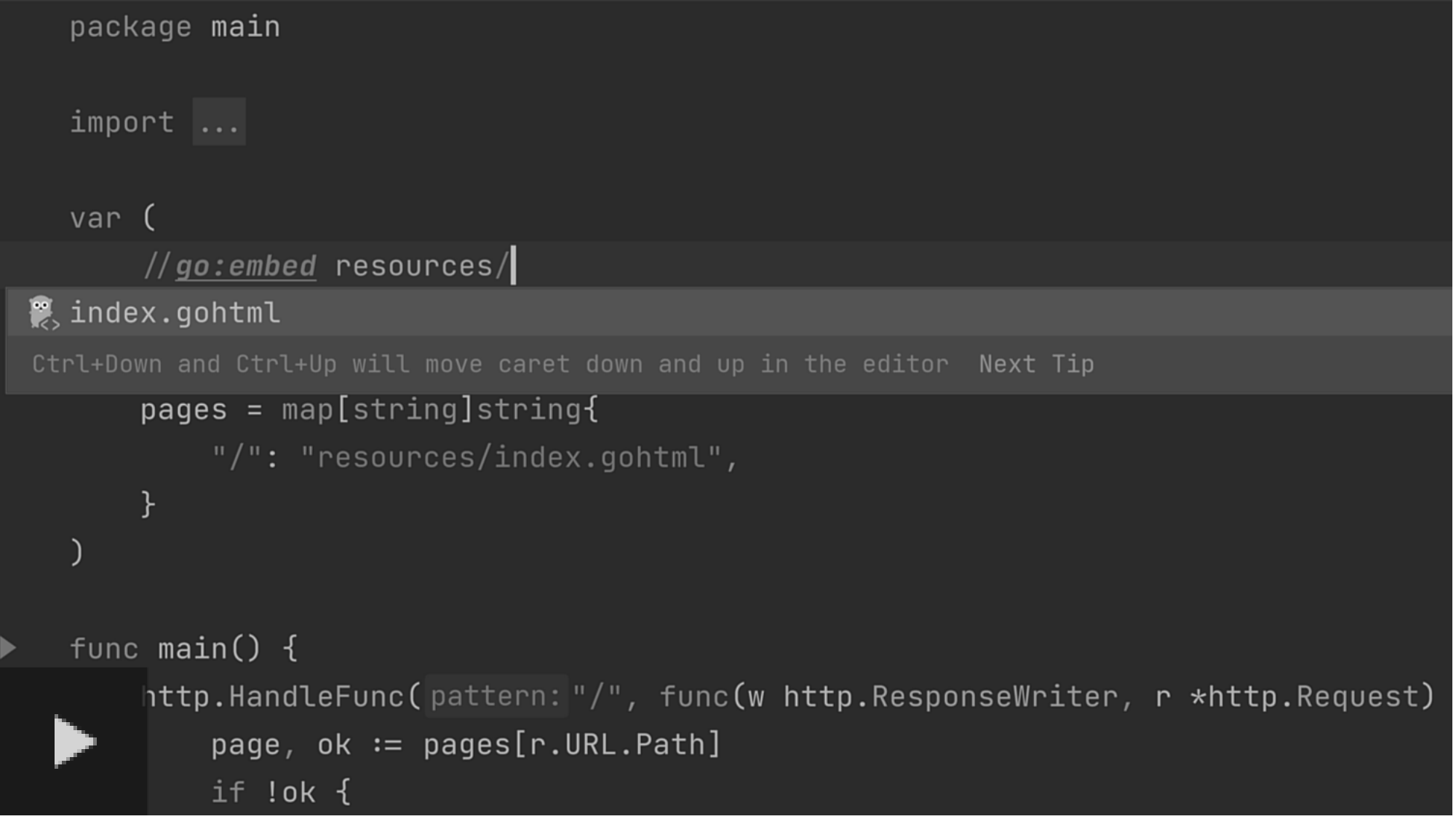
We'll start with the imports section, where we can see that we are using a new package called embed. This package, combined with the comment //go:embed, a compiler directive, tells the compiler that we intend to embed files or folders in the resulting binary.

You need to follow this directive with a variable declaration to serve as the container for the

embedded contents. The type of the variable can be a string, a slice of bytes, or embed.FS type. If you embed resources using the embed.FS type, they also get the benefit of being read-only and goroutine-safe.

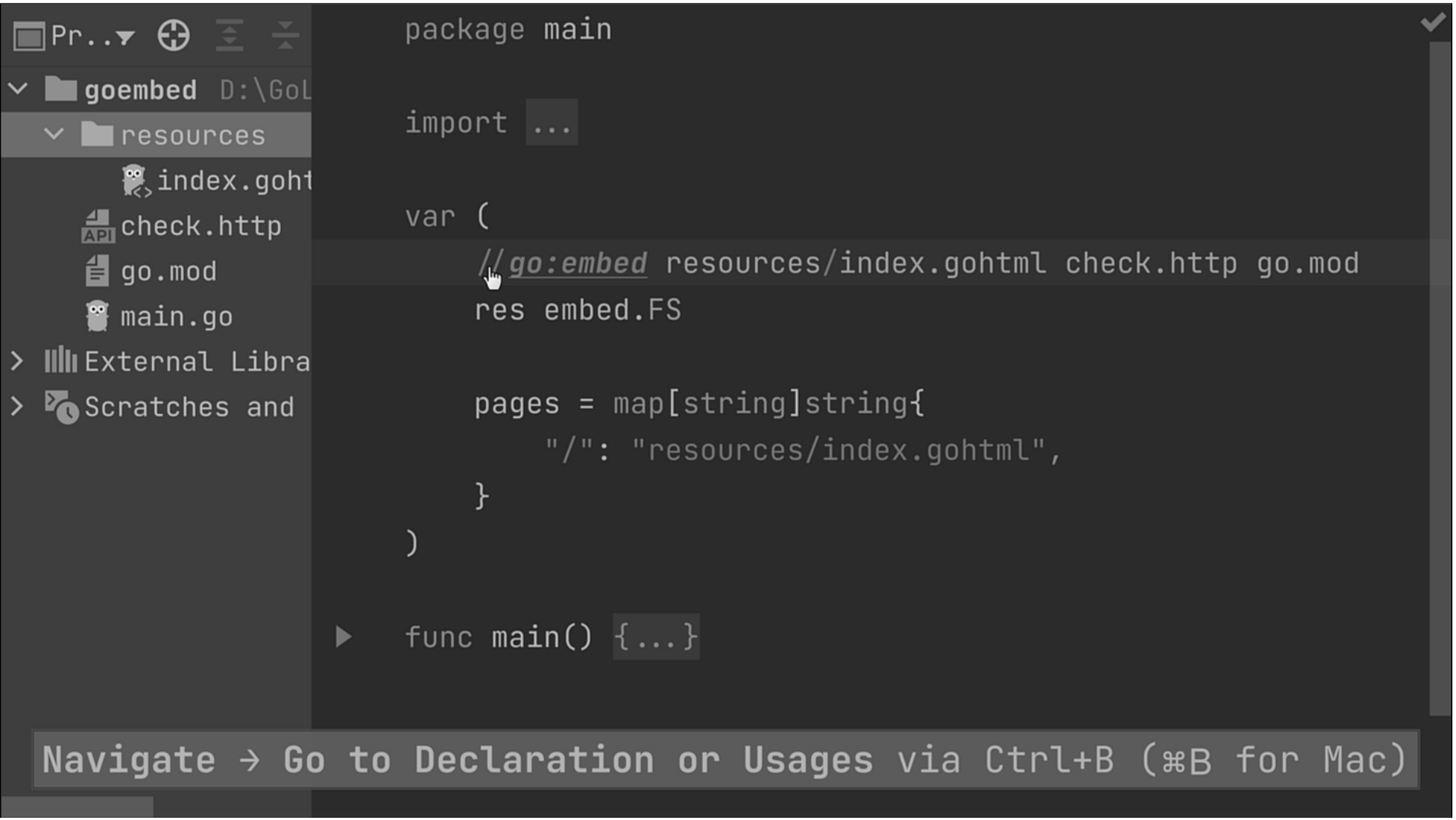
GoLand support for go:embed

GoLand completion features come in handy while using the embed directive, helping you write the paths/pattern.

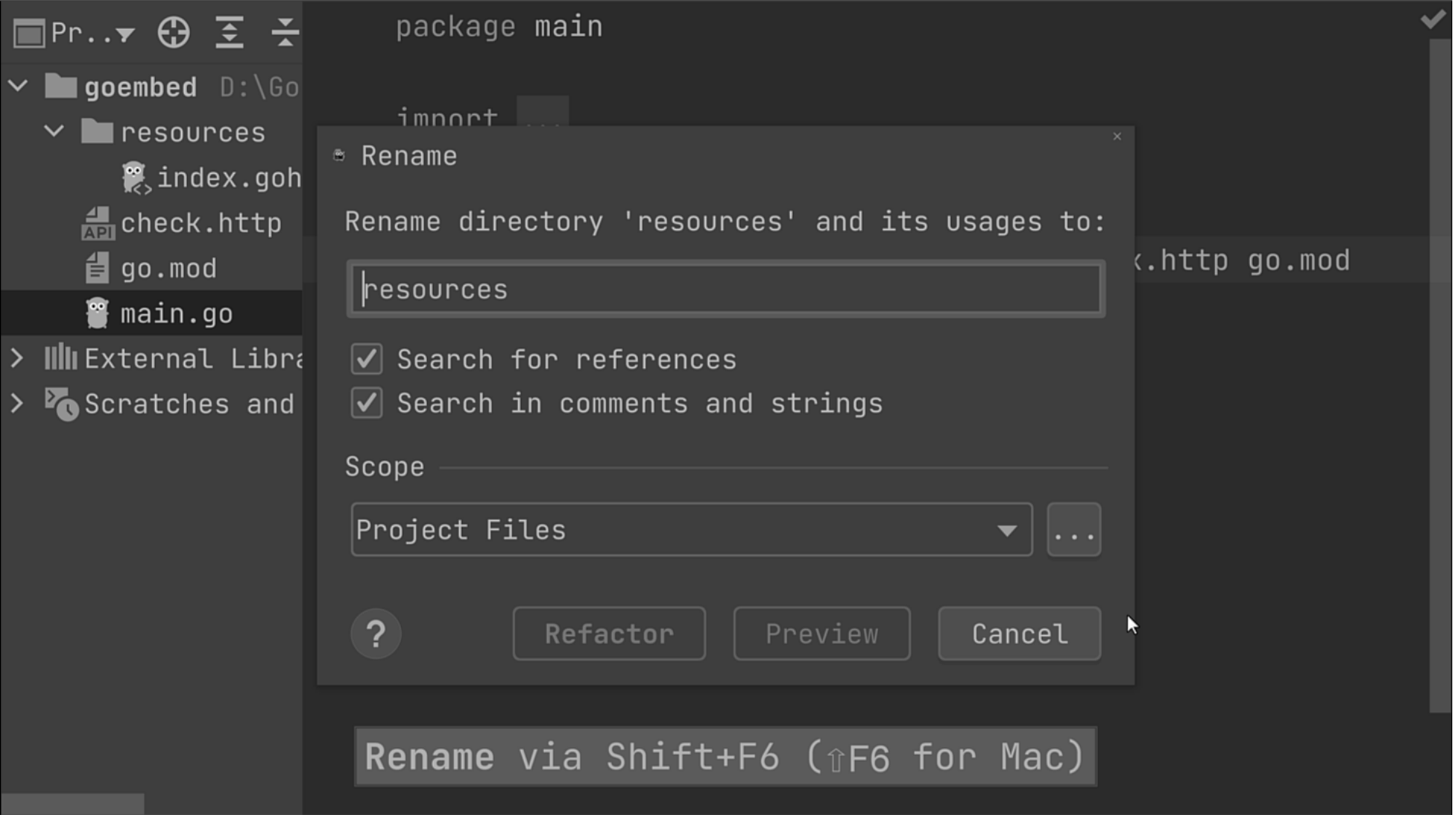


You can also navigate to the embedded resource

from the editor.



What if you want to change the name of the resource you've embedded? Or perhaps you want to change the whole directory structure? GoLand has you covered here too:



Pro tip: You can embed resources into the binary from any file, not just the main one. This means that

you can ship modules with resources that are transparently compiled into the end application.

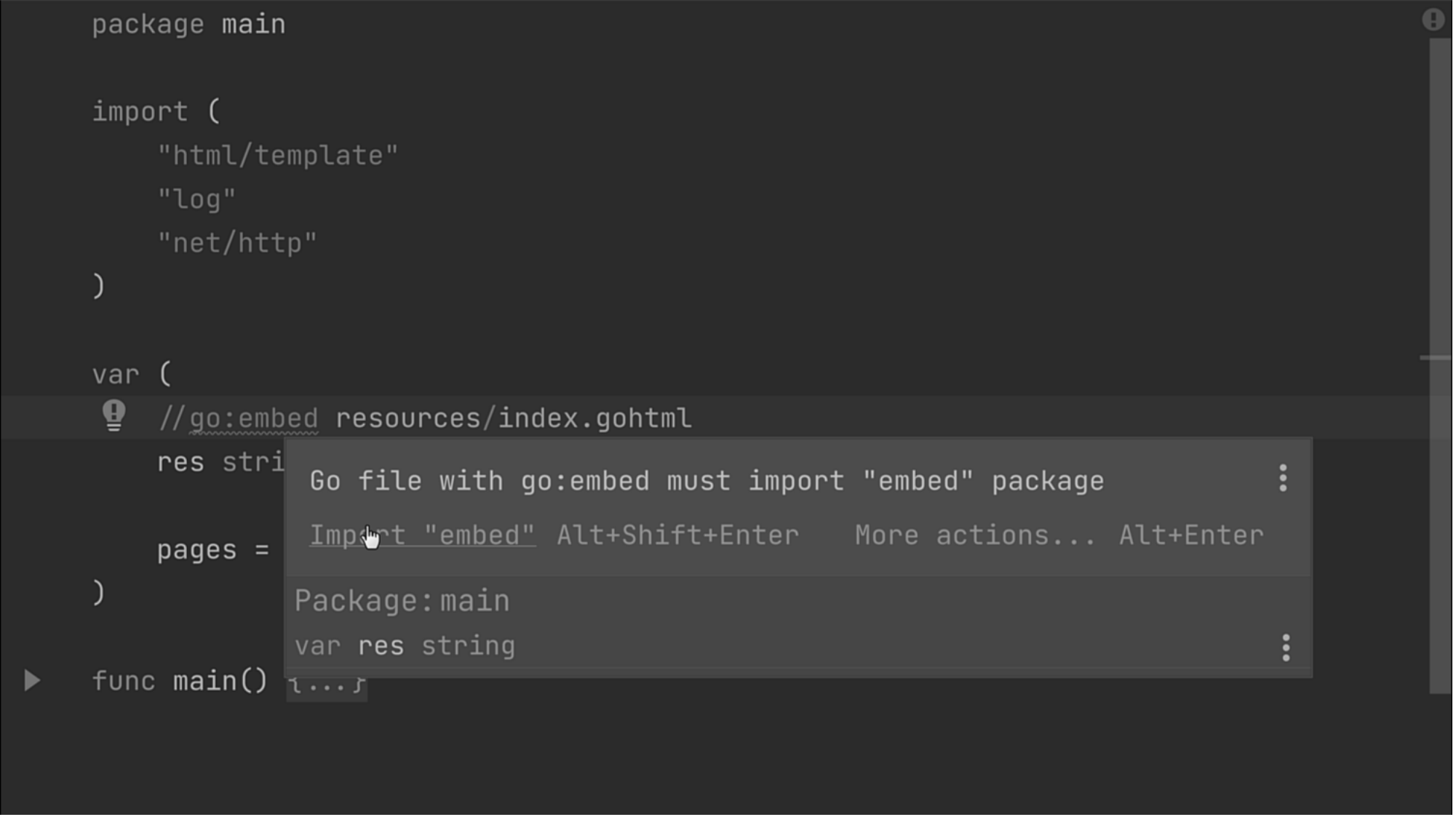
Pro tip: You can use the embedding feature in test files too! Try it out and let us know what you think.

Limitations

Embedding empty folders is not supported. Also, it's not possible to embed files or folders that start with "." or "_". This will be addressed in an upcoming version of Go, thanks to this related issue.

Embedding symlinks is not currently supported either.

If you don’t plan to use embed.FS, then you can use //go:embed to embed a single file. To do so, you must still import the embed package, but only for side effects.

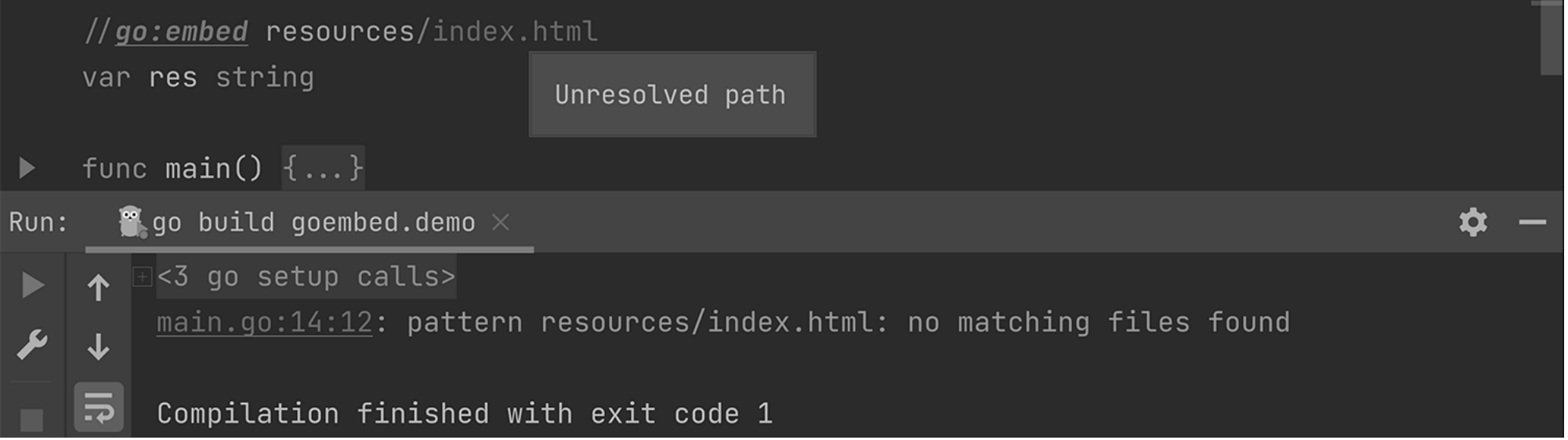


The embedding directive must not contain a space between the comment and “go:“.

```
// go:embed resources/index.gohtml
var res string

//go:embed resources/index.gohtml
var res string
```

The embedded paths must exist and match the pattern. Otherwise, the compiler will abort with an error.



Conclusion

That’s it for now! We learned why and how to use Go 1.16’s new embedding feature, took a look at how it works, and considered some caveats to remember when using it. We’ve also seen how

GoLand helps you work with this feature and provides features such as completion, error detection, and more.

We are looking forward to hearing from you about how you use this feature. You can leave us a note in the comments section below, on Twitter, on the Gophers Slack, or our issue tracker if you'd like to let us know about additional features you'd like to see related to this or other Go functionality.